CONTRIBUTED PAPER

The Development of Turbo and LDPC Codes for Deep-Space Applications

Error-correcting codes that allow very power-efficient information transmission from spacecraft have been developed along with suitable encoding and decoding hardware.

By Kenneth S. Andrews, *Member IEEE*, Dariush Divsalar, *Fellow IEEE*, Sam Dolinar, *Member IEEE*, Jon Hamkins, *Student Member IEEE*, Christopher R. Jones, *Member IEEE*, and Fabrizio Pollara, *Member IEEE*

ABSTRACT | The development of error-correcting codes has been closely coupled with deep-space exploration since the early days of both. Since the discovery of turbo codes in 1993, the research community has invested a great deal of work on modern iteratively decoded codes, and naturally NASA's Jet Propulsion Laboratory (JPL) has been very much involved. This paper describes the research, design, implementation, and standardization work that has taken place at JPL for both turbo and low-density parity-check (LDPC) codes.

Turbo code development proceeded from theoretical analyses of polynomial selection, weight distributions imposed by interleaver designs, decoder error floors, and iterative decoding thresholds. A family of turbo codes was standardized and implemented and is currently in use by several spacecraft. JPL's LDPC codes are built from protographs and circulants, selected by analyses of decoding thresholds and methods to avoid loops in the code graph. LDPC encoders and decoders have been implemented in hardware for planned spacecraft, and standardization is under way.

KEYWORDS | Bounds; Consultative Committee for Space Data Systems (CCSDS); error-correction coding; low-density parity-check (LDPC) code; protograph; spacecraft; space vehicle communication; turbo code

Manuscript received January 22, 2007; revised April 11, 2007. This work was supported by the IND Technology Program, under contract with the National Aeronautics and Space Administration.

The authors are with the Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109 USA (e-mail: andrews@shannon.jpl.nasa.gov; dariush@shannon.jpl.nasa.gov; sam@shannon.jpl.nasa.gov; hamkins@jpl.nasa.gov; crjones@jpl.nasa.gov; fabrizio@shannon.jpl.nasa.gov).

Digital Object Identifier: 10.1109/JPROC.2007.905132

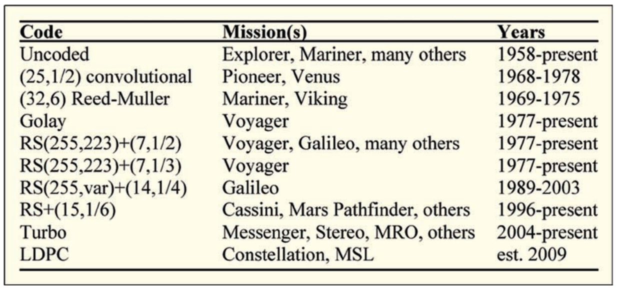
I. INTRODUCTION

In 1948, Shannon [1] proved that every noisy channel has a maximum rate at which information may be transferred through it and that it is possible to design error-correcting codes that approach this capacity, or *Shannon limit*, provided that the codes may be unbounded in length. For the last six decades, coding theorists have been looking for practical codes capable of closely approaching the Shannon limit.

For more than four decades, NASA and the Jet Propulsion Laboratory (JPL) have been sending deep-space probes to explore the far reaches of our solar system. Because of the extreme dilution of signal power over interplanetary distances, JPL has always taken more than an academic interest in searching for codes that approach Shannon's limit as closely as possible.

The saga of error-correcting codes for deep-space missions is summarized in Table 1. Initial missions in the late 1950s and early 1960s sent their data uncoded. By the late 1960s and early 1970s, missions were using codes of that era, such as Reed–Muller and long constraint-length convolutional codes (the latter decoded with a suboptimal sequential decoder). Voyager launched in 1977 with the state-of-the-art optimized (7, 1/2) convolutional code, to be decoded with a maximum-likelihood Viterbi decoder developed partly at JPL years earlier [2]. Voyager's engineers also anticipated needing an even stronger code for compressed data and for extended-mission visits to Uranus and Neptune, so they included an encoder for a (255, 223) Reed–Solomon code [3] to be concatenated with the convolutional code when needed. Concatenations of two relatively simple codes had been proposed by Forney in his thesis [4] a decade earlier as a way to create a powerful

Table 1 Codes Used by NASA Missions



Code	Mission(s)	Years
Uncoded	Explorer, Mariner, many others	1958-present
(25,1/2) convolutional	Pioneer, Venus	1968-1978
(32,6) Reed-Muller	Mariner, Viking	1969-1975
Golay	Voyager	1977-present
RS(255,223)+(7,1/2)	Voyager, Galileo, many others	1977-present
RS(255,223)+(7,1/3)	Voyager	1977-present
RS(255,var)+(14,1/4)	Galileo	1989-2003
RS+(15,1/6)	Cassini, Mars Pathfinder, others	1996-present
Turbo	Messenger, Stereo, MRO, others	2004-present
LDPC	Constellation, MSL	est. 2009

overall code with a decoding complexity equivalent to the complexity of the individual component decoders. This venerable Reed–Solomon and convolutional concatenated coding system has remained a standard in deep space and many other communication systems for the past three decades.

Despite this concatenated code's success in deep space, JPL continued to look for codes that approached Shannon's limit even more closely. The popular wisdom in the coding community was that near-optimal codes could only be obtained with increased minimum distance and exponentially increasing decoding complexity. Deep-space applications, with their extremely expensive probes in space and a willingness to invest in complex ground systems, were one area where highly complex codes could prove worthwhile if they returned even a smidgen of extra coding gain. To this end, JPL conducted a search for an additional 2 dB of coding gain [5] in the 1980s. This search culminated in the selection of a (15, 1/6) convolutional code that was eventually launched with the Mars Pathfinder and Cassini missions in the mid-1990s, which spawned a round of research into efficient parallel architectures [6], [7] for the extremely complex Viterbi decoder required to support this code.

Galileo became an early test bed for the new coding schemes in early 1991 when its main antenna failed to unfurl on its way to Jupiter and the project faced a massive reduction in planned data rate. The coding group at JPL proposed and implemented software encoding and decoding of a concatenated code with a variable-redundancy version of the standard Reed–Solomon outer code and a complex (14, 1/4) inner convolutional code. We further proposed "redecoding" the convolutional code several times based on partial solutions found by the Reed–Solomon decoder. The redecoding idea and the method to modify a Viterbi decoder to pin a subset of known trellis states based on prior Reed–Solomon decoder output had been suggested by other researchers [8]–[11]. Galileo ultimately implemented for its mission at Jupiter a feedback concatenated decoder [12] that iteratively decoded the concatenated code using four stages of feedback between inner and outer decoders [13], [14], improving on a single-pass decoder by more than 0.5 dB. JPL had tiptoed into the modern realm of iterative decoding but did so along a path that quickly turned out to be an evolutionary dead end, a *Neanderthal* to the *homo sapiens* emerging on the world stage.

A relatively unknown trio of researchers, Berrou, Glavieux, and Thitimajshima, profoundly changed the coding world with their introduction of turbo codes in 1993 [15], a simple but clever parallel concatenation of two convolutional codes that achieved dramatically larger coding gains than any other codes of the day, and at substantially lower complexity. The key insights were the introduction of an interleaver between the two convolutional codes, and an iterative suboptimal decoder that passes information back and forth between two relatively simple maximum a posteriori probability decoders. The claimed performance was so good that the initial reaction of the coding establishment was deeply skeptical.

Researchers at JPL were among the first, along with [16]–[19], to analyze and verify [20]–[22] the turbo code claims, and to extend the concept of turbo codes—from two constituent codes to multiple codes [23] to more general trellises [24], from a parallel concatenation to a serial concatenation [25], and so on. Before Cassini launched in 1996 with its high-complexity convolutional code, JPL had already begun standardization of turbo codes for future space missions.

In 1998, at about the time deep-space missions were signing up to use turbo codes, MacKay visited JPL to present a talk entitled "Making Gallager Codes that Beat Turbo Codes." He showed that *low-density parity-check* (LDPC) codes, originally introduced in Gallager's thesis in 1960 [26], can be designed to perform as well as, or better than, turbo codes.

The rediscovery of LDPC codes led to more coding research at JPL and around the world. LDPC codes had more degrees of design freedom compared to turbo codes, which enabled designers to more effectively trade off threshold and error floor performance or other attributes. On the other hand, this increased flexibility meant that many of the early LDPC designs were unstructured, leading to impractical decoding—and even encoding—complexities. A key insight enabled a structured analysis and design of LDPC codes based on *protographs* [27], and eventually JPL's high-speed decoders.

The renaissance of LDPC codes did not mark the end of turbo codes, however. LDPC codes have performance and complexity advantages over turbo codes at high code rates, but turbo codes are currently still the best solution for the lower code rates. This natural partition meant that the standard family of turbo codes at rates 1/6, 1/4, 1/3, and 1/2 [28] could live in harmony with a proposed standard of LDPC codes at rates 1/2, 2/3, 4/5, and 7/8 [29].

This paper describes the turbo codes and LDPC codes developed for deep-space applications. It includes discussions of designs, implementations, performance, and standardization.

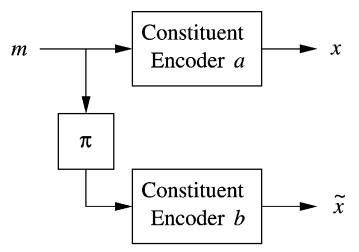


Fig. 1. *A turbo encoder.*

II. TURBO CODES

Turbo codes are constructed by applying two or more simple-to-decode encoding rules to different permutations of the same information sequence. The turbo encoder in Fig. 1 applies encoder a to the information sequence m in its initial ordering and encoder b to the permuted information sequence $\pi(m)$. Both outputs x and $\tilde{x}$ from the parallel branches of the encoder are concatenated to form turbo codewords. Short constraint-length convolutional codes are often chosen for the constituent components, one of which is usually chosen to be systematic. The resulting turbo code is also known as a parallel concatenated convolutional code.

The received noisy symbols y and $\tilde{y}$ are decoded iteratively, as shown in Fig. 2. There is a simple decoder for each of the component codes that makes probability estimates or *soft decisions* on each of the message bits. Each soft decision is decomposed into an *intrinsic* component that is due to the noisy observation of that bit and an *extrinsic* component that represents information from the adjacent bits. The extrinsics are permuted and exchanged as *a priori* probabilities among the constituent decoders. Decoding proceeds iteratively according to a message passing schedule until the constituent decoders reach a consensus $\hat{m}$ on the original message bits m.

A. Design and Construction

As for all linear codes, a good turbo code design assures that every nonzero codeword has a sufficiently large number of nonzero symbols, or *Hamming weight*. By linearity, this assures that every codeword is sufficiently different from any other that the probability of mistaking one for the other is small. At high signal-to-noise ratio (SNR), this *minimum distance* of a turbo code determines its error rate, and so good code design techniques assure that the minimum distance grows with the code length.

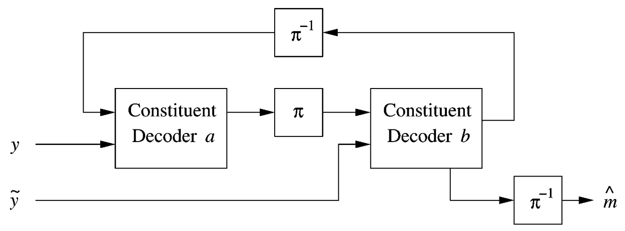


Fig. 2. *A turbo decoder primarily contains a feedback loop with two modified BCJR decoders, a permutation π, and its inverse.*

Good turbo codes are constructed using short constraint length, infinite impulse response (IIR) convolutional codes as components. The IIR property is important because a single isolated information bit error will produce a large encoded weight no matter what permutation is used. On the other hand, information sequences corresponding to two or more bit errors can be permuted to different bit patterns whose encoded output bears no resemblance to the encoding of the unpermuted information. The trick in turbo coding is to match low-weight encodings of one permutation with high-weight encodings of the other(s), thus producing total weights significantly higher than the low minimum weights that characterize the simple component codes.

1) Trellis Termination: Following the k information bits, an additional t tail bits are appended in order to drive the encoder to the all-zero state at the end of the block. This action of returning the encoders to the all-zero state is called *trellis termination*. Due to the encoder's recursiveness, the required t tail bits cannot be predetermined but they can be automatically computed at the encoder using a trick suggested in [30] and illustrated by the switches shown in Fig. 11.

2) Choice of Convolutional Code Polynomials: Early researchers tried to improve on the codes of Berrou *et al.* by increasing the constraint length of the constituent codes. However, it was soon discovered that constraint lengths higher than Berrou *et al.*'s original choice of five led to decoders that did not converge as effectively and hence required higher thresholds. Better constituent codes did not necessarily produce better turbo codes.

The first improved design criterion resulting from analysis was to select a primitive feedback polynomial for the recursive convolutional component code, instead of a nonprimitive polynomial as used in the original incarnation of Berrou *et al.* The encoded weights of self-terminating weight-2 input sequences increase linearly with the distance between its two 1s, with a slope that depends on the characteristics of the constituent codes. When they are far apart, the encoded sequence looks like the code's infinite impulse response up to the point where the second bit error terminates the further accumulation of weight. With a primitive divisor polynomial, all weight-2 inputs are guaranteed to cause a traverse through all of the trellis states before any possible return to the all-zero state, piling up maximum encoded weight. Still, it is the job of the permuter to further ensure that these minimal-length traverses in one component code's state diagram are paired with longer traverses in another component with differently permuted input.

3) Weight Distributions and Interleaver Designs: In an early paper [31], we looked at the weight distributions achievable for turbo codes using random, nonrandom, and semirandom permutations. With recursive encoders, non-self-terminating sequences have little effect on decoder performance because they accumulate high encoded weight until they are artificially terminated at the end of the block. From probabilistic arguments based on selecting the permutations randomly, it was concluded that self-terminating weight-2 data sequences are the most important consideration in the design of the constituent codes. Higher weight self-terminating sequences have successively decreasing importance. These considerations outweigh the criterion of selecting component codes that would produce the highest minimum distance if unpermuted.

Random permutations do a very good job of teaming low weights with high weights for the vast majority of possible information sequences. However, it is still highly likely that a few two-bit-error sequences with low encoded weights will be randomly permuted into other two-bit-error sequences with low weights. It is possible to design nonrandom permutations that ensure that the minimum distance due to weight-2 input sequences grows roughly as $\sqrt{2k}$, where k is the information block length. However, the nonrandom permutations most optimized with respect to weight-2 inputs amplify the bad effects of higher weight inputs, and as a result they are inferior in performance to randomly selected permutations. But there are "semirandom" permutations that perform nearly as well as the designed nonrandom permutations with respect to weight-2 input sequences and are not as susceptible to being foiled by higher weight inputs.

Similar properties have been built into algorithmic interleavers, including, most notably, the original interleaver proposed by Berrou [16] and the more recent and even simpler quadratic interleaver [32].

B. Implementation

1) Turbo Encoders: A turbo encoder as shown in Fig. 1 can be implemented directly in hardware or software and is fast and simple. The only particular design decision is in implementation of the interleaver, either as a table in memory, or as a small algorithm.

2) Turbo Decoders: Decoding is more difficult. As shown in Fig. 2, a decoder applies a modified Bahl–Cocke–Jelinek–Raviv (BCJR) algorithm [33] to the noisy symbols y from one of the constituent convolutional codes, applies permutation π to the resulting "extrinsic information," combines the result with the noisy symbols $\tilde{y}$ from the other constituent code, inversely permutes the resulting extrinsic information, and repeats. After some number of iterations, an estimate $\hat{m}$ is made of the transmitted message. The BCJR algorithm, also known as the "forward–backward" algorithm, is moderately complex and inherently serial. While a codeword can be broken into overlapping windows on which the BCJR algorithm can be performed independently and potentially in parallel, this makes the sequence of operations yet more complex. For this reason, turbo decoders are more readily implemented in software on a general-purpose processor than in application-specific hardware. There are exceptions, such as a remarkable hardware decoder built by Sony [34] that uses 200 copies of a custom application-specific integrated circuit (ASIC) and has implementation losses under 0.03 dB. As with the LDPC decoding algorithm described later, the BCJR equations can be formulated in a variety of ways. Most implementers choose to use "min-star" operations in the log-likelihood domain, rather than sums and products in the probability domain or another choice, for their simplicity and stability when values are quantized to small integers.

The turbo decoders in the Deep Space Network (DSN) are implemented on Texas Instruments 320C6203 digital signal processors (DSPs), with the key algorithms written in hand-optimized assembly language for speed [35]. Each decoding system contains 16 DSPs: 14 for turbo decoding, one for frame synchronization, and one for scheduling, control, and status. Each decoder DSP operates independently from the others and is assigned decoding tasks by the scheduler as it matches arriving noisy codewords to idle DSPs.

A stopping rule is used to reduce the average number of iterations required. The intent of the stopping rule is to evaluate the decoder's tentative result at the end of each iteration and decide if it is sufficiently likely to be correct that further iterations are not necessary. Researchers have proposed a variety of stopping rules, and the one implemented in the DSN is rule S_2 in [36] that compares the minimum absolute value of the decoded-bit reliabilities to a threshold. If the estimated reliability of any message bit fails to exceed the programmed threshold, the decoder stops after some maximum number of iterations is reached. As the threshold is reduced from ∞ (at which the maximum number of iterations is performed on every frame), fewer iterations are performed on average and the error rate increases. Example performance curves are shown in Fig. 3 for a maximum of ten iterations. It is evident that decoder speed can be increased considerably with virtually no loss in performance, after which there is a tradeoff between speed and performance.

When using a stopping rule, codewords take variable amounts of time to decode, and with several decoders running in parallel, they may be released out of sequence. In the DSN implementation, a large buffer is included in which the codewords are reordered. Additional logic tracks codeword arrival times accurate to 100 ns [37], along with other metadata.

Turbo codes have been included on several spacecraft. A test of turbo codes was performed in late 2004 by SMART-1, a lunar orbiter, but they were not used for significant data

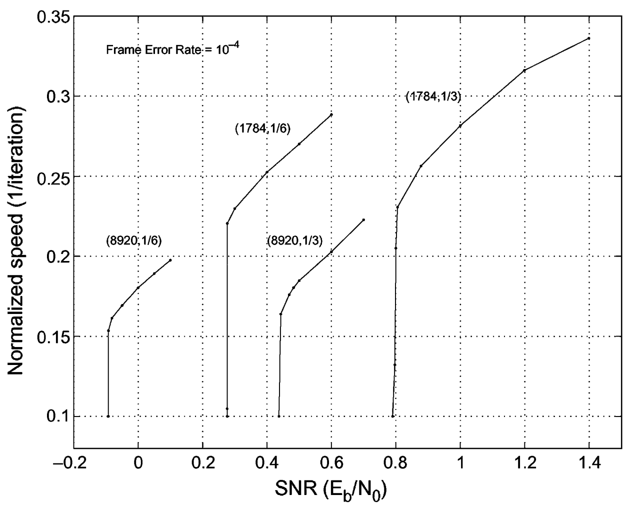


Fig. 3. ***Average decoder speed (in codewords/iteration) as a function of SNR as the stopping rule threshold is varied for four turbo codes with information lengths $k = 1784$ and $k = 8920$ and code rates 1/3 and 1/6.***

return. The MESSENGER spacecraft (a Mercury orbiter launched on August 3, 2004), Mars Reconnaissance Orbiter, and New Horizons (Pluto flyby) are all currently using turbo codes for their primary data return.

III. LOW-DENSITY PARITY-CHECK CODES

LDPC codes were first introduced by Gallager [26] in his 1960 Ph.D. dissertation. Initially, they were prohibitively complex and were subsequently forgotten until a series of papers from MacKay [38], [39] in the late 1990s generated renewed interest.

Any LDPC code can be described via its low-density parity-check matrix containing only a few ones in each row and column. Alternatively, they have a graphical representation introduced by Tanner in 1981, and these graphs also describe the routing for a message-passing decoding algorithm. Tanner graphs are bipartite, with each edge connecting a *variable node* to a *check node*. Each check node corresponds to a constraint in the parity-check matrix, and each variable node connected to the channel corresponds to a coded symbol.

Gallager's original LDPC codes were *regular*, in that all variable nodes had the same degree (number of connected edges), as did all check nodes. With the rediscovery of LDPC codes in the past decade, it was found that irregular degree distributions can be designed to improve the performance for very large blocks.

A. Design and Construction

As the graph describing the code grows, connections in a randomly organized graph can become difficult to manage at high speed in a dedicated piece of hardware. To alleviate this problem, with little or no loss in observed performance, structured graph designs have been proposed. Two similar structured graph approaches are multiedge-type graphs [40] and protographs [27], [41]. Quasi-cyclic LDPC codes [42] are an important special case.

1) Protograph Designs: A structured LDPC code can be constructed by taking a small Tanner graph, or *protograph*, replicating it many times, and interconnecting the copies as shown in Fig. 4. When the protograph is replicated T times, each edge is replicated into a bundle of T edges, now connecting T variable nodes to T check nodes. The copies of the protograph are interconnected by "unplugging" these edges from their check node sockets, permuting them, and reconnecting them. This process is repeated for each bundle of T edges, or *edge type*.

While the resulting *derived* or *lifted* graph is T times as large as its protograph, it inherits many of the protograph's properties. It has the same code rate (except possibly for coincidental redundancies due to the particular permutations chosen) and the same distribution of variable and check node degrees. Moreover, neighborhoods are preserved: in each of the graphs in Fig. 4, for example, every degree-1 variable node connects to a degree-3 check node,

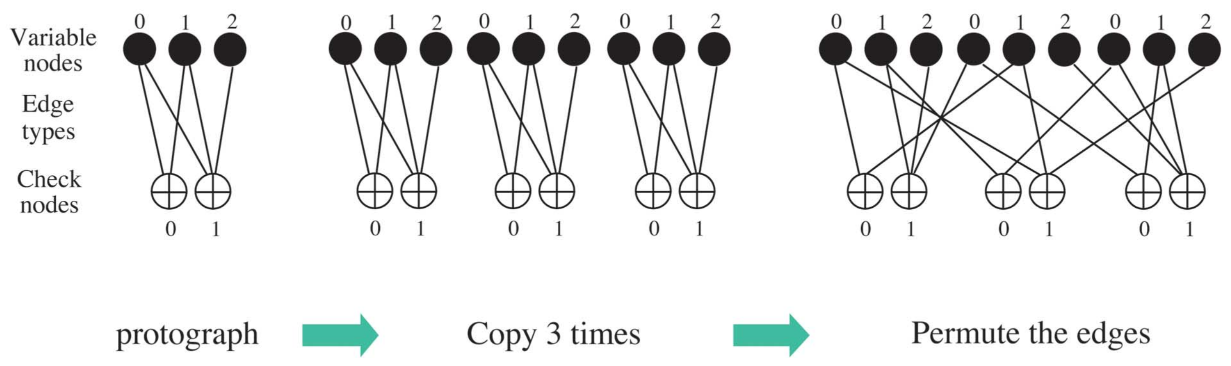


Fig. 4. ***A protograph gives rise to a larger lifted graph.***

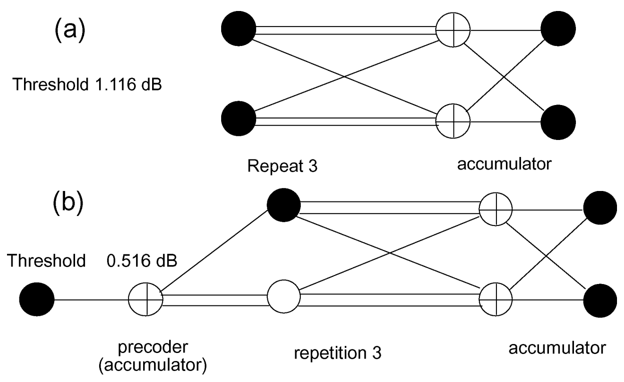


Fig. 5. ***Protographs for (a) a systematic rate-1/2 RA code and (b) a rate-1/2 ARA code.***

which is also connected to two degree-2 variable nodes, and so on. These properties mean that full LDPC codes can be designed by applying techniques such as density evolution [59]–[62] to the protograph.

A rate-1/2 systematic repeat-accumulate (RA) code with repetition-3 is shown in Fig. 5(a). The threshold SNR required for reliable decoding can be dramatically improved by "precoding" the repetition code with an accumulator [43], as shown in Fig. 5(b). To preserve the rate of the resulting *accumulate-repeat-accumulate* (ARA) code, one of the variable nodes is punctured (not transmitted), as indicated by the open circle in the figure.

In an ARA code protograph, the number of degree-2 variable nodes is equal to the number of inner checks (checks that are connected to these degree-2 variable nodes), and it turns out that the minimum distance of the resulting codes grows slowly as the block length n is made large. Instead, if some of the degree-2 variable nodes are replaced with higher degree nodes, the minimum distance of the resulting codes can be made to grow linearly with n, with high probability [44]. An example of such an AR4JA ("Accumulate Repeat-4 Jagged-Accumulate") protograph is shown in Fig. 6, along with a way of adding variable nodes to increase the code rate. The rate-1/2 protograph in this family has a decoding threshold of $E_b/N_0 = 0.62$ dB.

2) Construction of Protograph Codes: A protograph is expanded or lifted into a full code graph by the design of a permutation of size T for each edge in the protograph. Popular design techniques use either structured permutations, such as cyclic shift permutations called *circulants*, or random-like permutations generated by computer search with an optimization criterion. Both techniques aim to avoid small loops in the code graph because loops introduce dependence between random variables that is unaccounted for in the belief propagation algorithm. Belief propagation determines exact a posteriori probabilities on a graph that contains no loops [45], but in the presence of more than one loop [46], the algorithm is only approximate and it can even fail to reach any answer. A collection of small loops can also form a *stopping set*, a problematic graph structure when a belief propagation decoder is used with the binary erasure channel.

Among computer search methods, progressive edge growth (PEG) [47] is a greedy algorithm that sequentially inserts graph edges each step in a location that maximizes the minimum loop length, or girth, of the resulting graph.

A graph with large girth is free from small loops, but other metrics better indicate the presence of either small loops or stopping sets. The extrinsic message degree (EMD) [48] of a variable node set counts the number of check nodes connected exactly once to that set. While EMD is computationally burdensome, the approximate cycle EMD (ACE) [48] metric is a practical substitute that counts only the number of check node connections to each loop in a graph. Evaluation of ACE has complexity linear in code block length, and its use can reduce error floors by two to three orders of magnitude from randomly constructed graphs.

The LDPC codes described in Section V were built from the protographs of Fig. 6 in two stages. Each protograph was first lifted by a factor of $T = 4$ with PEG (yielding 20 variable nodes, 16 of them transmitted) to eliminate parallel edges. The resulting graph was then lifted by a factor of $T = n/16$ using circulants with "phases" selected by a Viterbi-like algorithm from [48] that aimed to maximize a variant of ACE, with an additional cost for particularly small loops.

B. Implementation

1) LDPC Encoders: An LDPC code is specified by its sparse parity check matrix H, unlike a turbo code that is specified by its encoder circuit. There are various ways to build an encoder that maps information sequences of length k into codewords of length n that satisfy the $n - k$ given linear constraints. The obvious method is to invert H to find the systematic generator matrix G, and then to perform encoding by matrix multiplication. Except in special cases, G is dense, and hence encoding complexity is

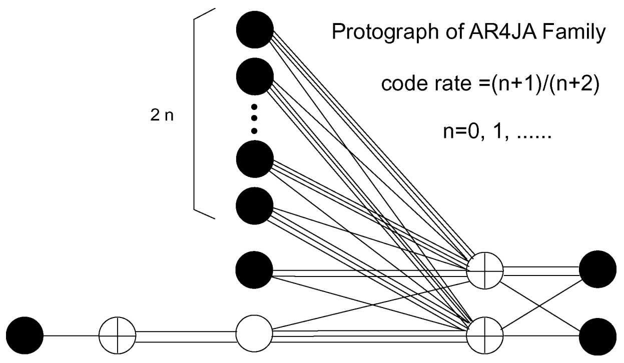


Fig. 6. ***Protographs in the AR4JA family with rates 1/2 and higher.***

quadratic in the codeword length. Those special cases include RA and irregular repeat-accumulate (IRA) codes [49], which is one reason these constructions have attracted particular interest. Unfortunately, this simple encoding method requires that at least $n-k$ variable nodes have degree ≤ 2, and this comes at a cost in decoding threshold and minimum distance.

In a landmark paper, Richardson and Urbanke demonstrated that by using back-substitution, one can build encoders for most LDPC codes with complexity that grows almost linearly in block length [50]. While a remarkable result, this did not solve the encoding puzzle because the resulting computations require a large matrix that is sparse but otherwise largely disorganized. This matrix must be stored in the encoder, which can be a burden in the deep-space environment. Their encoding algorithm is also essentially serial, limiting the speed at which such encoders can run.

Some block-circulant parity check matrices have block-circulant matrix inverses. While the resulting generator matrices are dense, they are highly structured. Encoders based on this fact consist of a set of shift registers and are both fast and fairly simple [51], [52].

2) LDPC Decoders: An LDPC decoder is provided with quantized soft symbol metrics at each variable node in the code graph. An edge metric is associated with each edge in the graph, and these are initialized to zero before decoding. Decoding proceeds by alternately updating the edge metrics according to a computation performed by each variable node, and then by updating them again according to computations performed at each check node. More precisely, each variable node i is given a log-likelihood ratio (LLR) $\lambda_i = \ln(P(c_i = 0)/P(c_i = 1))$, indicating the a priori relative probability between the two binary possibilities for the code symbol c_i. Variable node i makes a soft decision about c_i, based on λ_i and the opinions $\{u_j\}$ of the associated check nodes, and passes the new information to each check node as the message v_j. Each check node then looks for consistency among the incoming edge messages v_j and returns its opinion in the form of an updated message u_i. Those equations are typically stated in log likelihood ratio or "reliability" form as

$$v_i = \lambda_i + \sum_{j\neq i} u_j \tag{1}$$

$$u_i = 2\tanh^{-1}\left(\prod_{i\neq j} \tanh\frac{v_i}{2}\right). \tag{2}$$

An LDPC decoder must simply perform these two operations a great many times, as fast as possible. This can be done on a general-purpose microprocessor, but this problem can be more efficiently solved with an ASIC or field-programmable gate array (FPGA). A typical FPGA implementation performs several variable node updates per clock cycle until all are updated, then several check node updates per clock cycle until all check nodes are updated, and then repeats for the next iteration. The number of updates per clock cycle is determined by the hardware resources available and is typically far fewer than the number of nodes in the graph.

Equation (1) is readily implemented but (2) is inconvenient. Instead, computation of (2) can be performed via a series of applications of the following two-input function

$$\begin{aligned}\min{}^*(v_1, v_2) = \operatorname{sgn}(v_1)\operatorname{sgn}(v_2)(\min(|v_1|, |v_2|) \\ + \ln\left(1 + e^{-(|v_1|+|v_2|)}\right) - \ln\left(1 + e^{-||v_1|-|v_2||}\right)\Big).\end{aligned} \tag{3}$$

This min* function consists of a min operation and two additive correction terms. The correction terms can be approximated with a lookup table, a two-segment linear approximation [53], or discarded entirely for a loss of up to 1.0 dB in required SNR. When v_i's are quantized, (3) remains commutative but is no longer associative, so the order of operations can be significant.

Equation (2) can also be written

$$\tilde{u}_i = \prod_{i\neq j} \operatorname{sgn}(v_i) \sum_{i\neq j} \tilde{v}_j \tag{4}$$

where $\tilde{v}_j = -\ln\tanh(|v_j|/2)$. Here, $\tilde{v}_j$ represents the unreliability of message v_j rather than its reliability [54]. Equations (1) and (4) are easy to implement, but care must be used in the transformation between reliabilities and unreliabilities.

Hardware LDPC decoders are most often implemented using (1) and (3), though Richardson has chosen (1) and (4) in [55], and other methods have been used.

A hardware "universal" decoder suitable for decoding unstructured graphs was also constructed at JPL and used to test hundreds of candidate code designs for threshold and error-floor performance at throughputs close to 10 Mbps on a Xilinx Virtex-II XC2V 8000 FPGA. One particular challenge associated with such a decoder is with the routing difficulty that can be posed by a large unstructured parity check matrix. A decoder suitable for decoding unstructured graphs was devised by partitioning the code's parity matrix into L macro columns and L macro rows as shown in Fig. 7 (for $L = 4$). The nonzero entries in each of these macro columns or rows represent a graph edge that carries an extrinsic LLR from a constraint node to a variable node or vice versa. Along with a crossbar that allows any macro column to be connected to any macro row, the schedule suggested in the figure has like-labeled data moving from row-to-column or column-to-row ordering without collision. A decoder based on such an

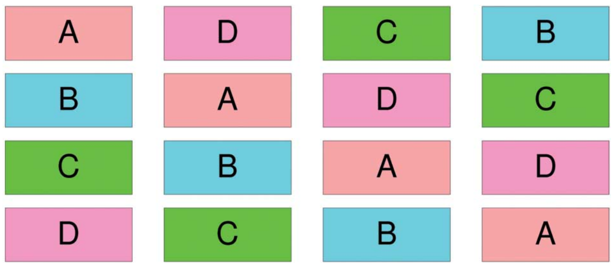


Fig. 7. *Collision-free interleave scheduling for an $L = 4$ parallel decoding implementation. Regions labeled with the same letter are interleaved from/to during the same time span.*

interleaver is able to "universally" decode any bipartite graph up to some complexity. However, because crossbar complexity scales as L^2, practical designs (and therefore the achievable parallelism) have been limited to $L = 16$.

For LDPC codes composed of circulants, fast hardware decoders can be built that take advantage of this structure. The decoder can update all the variable nodes (or check nodes) in one copy of the protograph in one clock cycle and use simple counters to track the circulant-expanded copies of the protograph. A faster decoder may update several copies of the protograph simultaneously. A smaller and slower decoder may update only a portion of the protograph in each clock cycle. Each of these decoders has been built at JPL in Xilinx FPGAs, giving decoder speeds that range from 2 to 80 Mbps, with a proportional range in FPGA resources consumed. The small 2-Mbps decoder is intended for implementation on the Mars Reconnaissance Orbiter for communication with the Mars Science Lander; the faster decoders may be used by future missions such as the Constellation Program [56].

3) LDPC Decoder Stopping Rules: Just as with turbo codes, a stopping rule can be used to reduce the number of iterations required. Because an LDPC decoder naturally seeks to identify the transmitted codeword (rather than the transmitted message as a turbo decoder does), it is natural to stop the decoder when its tentative answer is in fact a codeword. The probability of finding an incorrect codeword is determined by the code's minimum distance. A good LDPC code can have a large minimum distance, and this probability can be less than 10^{-10} at any E_b/N_0. Hence the codeword criterion can be used to detect decoder errors in addition to serving as a stopping rule. The undetected error rate is small, as it is with Reed–Solomon decoders, and unlike turbo decoders.

When a stopping rule is used, the number of iterations required is reduced from the worst case value to very nearly the average value, but the decoding time becomes variable. For example, when the AR4JA ($n = 8192$, $k = 4096$) LDPC code that JPL has proposed for Consultative Committee for Space Data Systems (CCSDS) standardization is used with additive white Gaussian noise (AWGN) at 1.35 dB (an operating point where a decoder not constrained by speed would achieve Word Error Rate (WER) less than 10^{-6}), the decoder requires an average of 22.5 iterations, but successful decodings of some codewords require 50 to 200 iterations with nonnegligible frequency.

Because the decoding time is variable, an input buffer must be provided between the decoder and a continuous received symbol stream from a spacecraft. Perhaps remarkably, this buffer can be small. Fig. 8 shows the system performance using buffers of sizes 0 to 5 frames, with frames arriving I_{arr} iterations apart, compared to the average iterations required ($I_{\text{avg}} = 22.5$), shown as the dashed vertical line. The upper curve shows the performance with no buffer, where the decoder can only perform as many iterations as are available between codeword arrivals. With a buffer that can store five noisy codewords, the decoder fails to decode those that require > 200 iterations but successfully decodes most of the others so long as the frame arrival rate is below the time required to perform the average 22.5 iterations.

IV. PERFORMANCE RELATIVE TO BOUNDS

A. Universal Bounds on All Types of Codes

Very long turbo and LDPC codes perform astonishingly close to capacity limits. Shorter codes perform farther from the capacity limits, but most of the difference is explained by finite-block-size constraints on the performance of any code. Using Shannon's sphere-packing bound (SPB) for the continuous-input AWGN (CI-AWGN) channel as a benchmark on the performance of optimal codes, we showed in 1998 [57] that well-designed finite-size turbo codes could approach this benchmark within a residual "imperfectness"

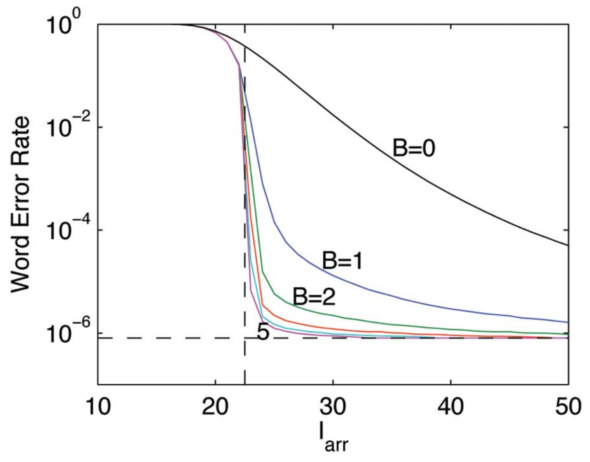


Fig. 8. *Performance of the variable-iterations decoder with preemptive buffer control as a function of frame interarrival time I_{arr}, with buffer sizes (right to left) $B = 0, 1, 2, 3, 4, 5$, AR4JA(8192,4096) code, $E_b/N_0 = 1.35$ dB.*

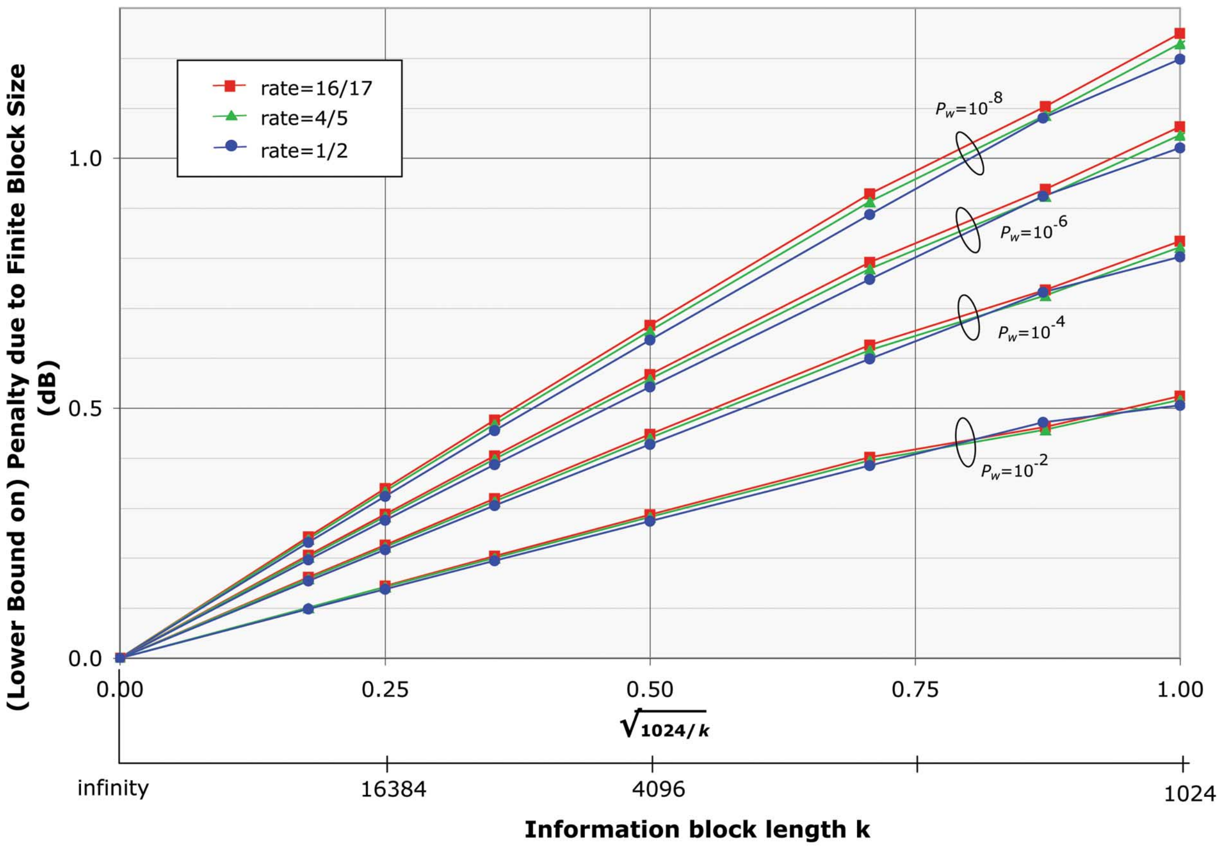


Fig. 9. ***Finite-size penalty obtained from the CI-AWGN SPB as a function of code rate r, information block size k, and codeword error rate P_w.***

of about 0.7 dB over a wide range of code sizes and rates up to 1/2. An approximation to the SPB for binary-input-constrained channels has proved to be a similarly useful benchmark for code rates higher than 1/2.

Our work on the SPB for the CI-AWGN channel uncovered a rather remarkable near-separation of the performance effects of finite-size constraints from those of rate and channel constraints that affect capacity calculations. For any channel and rate constraint, we can calculate the corresponding *capacity-constrained threshold* on E_b/N_0, above which arbitrarily low word error rate P_w is achievable in the limit as a code becomes arbitrarily large while maintaining the constrained rate. The SPB limits the achievable P_w if the code parameters are constrained to finite (n, k), or alternatively, it defines a *size-constrained optimal threshold* on E_b/N_0 that depends on (n, k, P_w). If we measure this size-constrained threshold as a function of (r, k, P_w), where $r = k/n$ is the code rate, the difference between the size-constrained threshold for fixed (k, P_w) and the capacity-constrained threshold for rate r is only weakly dependent on r, as illustrated in Fig. 9. Thus, a good rule of thumb is to use this difference as a measure of the *finite-size penalty* that should be added to the appropriate capacity-constrained threshold to approximate the size-constrained optimal threshold.

A given code's *size-constrained nonoptimality* at a prescribed P_w is the excess E_b/N_0 required for it to achieve error rate P_w over the corresponding size-constrained optimal threshold. Well-designed turbo and LDPC codes can reach a size-constrained nonoptimality of about 1/2 dB over a wide range of rates and sizes.

B. Union Bounds for Turbo Codes

In the first few years after the invention of turbo codes, union bounds [19], [22], [58] were calculated to characterize their performance, based on transfer functions derived from the state diagrams defining their recursive convolutional component codes. The premise for these bounds is the same as for the usual transfer function bounds applied to standard convolutional codes. The average error probability for an ensemble of turbo codes is upper bounded by a union bound that sums contributions from error paths of different encoded weights. This ensemble is defined by fixing the recursive convolutional component codes and allowing the interleaver(s) to be selected randomly (the *uniform interleaver* [19] assumption). The union bound on the ensemble average word error probability P_w or bit error probability P_b depends intrinsically on an *input–output weight enumerator* [19], [58], which counts the number of paths of each possible input–output weight combination. This is unlike the case of a plain convolutional code, for which the union bound on P_w depends only on the code's output weight enumerator. The ensemble average weight enumerator gives the *interleaver gain* [19] of the turbo code, which tells the rate at which P_w and P_b are lowered with increasing interleaver size k.

Union bounds proved useful for turbo codes because they accurately predict the level of the *error floor* of a turbo code's performance curve and suggest ways to design component codes and interleavers to lower this floor. The error floor is a low-slope region of the performance curve, wherein the turbo decoder's error rate decreases very slowly with increasing E_b/N_0. Above a certain E_b/N_0, the union bound basically tells the whole story, i.e., the P_w and P_b predicted by the bound are accurately achieved both by a maximum-likelihood decoder and by a turbo decoder, even though the latter is not a maximum-likelihood algorithm.

C. Iterative Decoding Thresholds for Turbo and LDPC Codes

The union bound calculations were useful for determining and explaining the performance of turbo codes in their error floor region and aided the design of good turbo codes to drive this floor as low as possible. The union bounds, however, did not predict the position of the "waterfall" portion of the turbo code's performance curve.

Density evolution analysis, introduced in the form of "EXIT charts" [59] or "extrinsic SNR" analysis [60]–[62], provided a method for obtaining an *iterative decoding threshold* for ensembles of turbo and turbo-like codes. Below this threshold, the iterative decoder fails with high probability, but above this threshold, the error probability can be driven to zero if the turbo code's interleaver is arbitrarily large and sufficiently randomized.

Density evolution was also an important tool for determining iterative decoding thresholds for ensembles of LDPC codes [63]. A landmark paper by Richardson *et al.* [64] determined and tabulated optimal irregular degree distributions yielding minimum thresholds for unstructured LDPC codes. Density evolution can also be applied to determine iterative decoding thresholds for structured LDPC codes, including multiedge-type codes [65] and protograph codes [27].

The iterative decoding threshold for protograph codes is a capacity-like limit in that it defines a minimum E_b/N_0 required for achieving arbitrarily small error rates in the limit as the protograph is expanded to build an arbitrarily large code. The difference between the protograph's iterative decoding threshold and the capacity limit for the protograph's rate is the *protograph nonoptimality* under iterative decoding. The protograph nonoptimality is solely a function of the small protograph and not of the size or the method by which it is expanded to build a large code. The remainder of a protograph code's size-constrained nonoptimality is its *expansion nonoptimality* that results from the particular choice of permutations to expand the protograph to a finite-size code.

Full density evolution is complex, so instead we have used a fast and accurate approximation to density evolution originally proposed in [66] as a protograph design tool. The reciprocal channel approximation makes use of a single real-valued parameter, the edge message SNR s, as a stand-in for full density evolution. For every value of s, a reciprocal, r, is defined such that $C(s) + C(r) = 1$, where $C(x)$ denotes the capacity of the binary-input AWGN channel with SNR x. In the reciprocal channel approximation, the parameter s is additive at variable nodes and the reciprocal parameter r is additive at check nodes. It is a simple matter to track the evolution with iterations of SNR messages $\overrightarrow{s}_e$ and $\overleftarrow{s}_e$ in both directions along all edges e of a small bipartite protograph. The protograph's threshold is the minimum value of the channel input SNR for which unbounded growth of all such edge messages is achieved.

Our best protograph code designs have aimed for a protograph nonoptimality around 0.4 dB. With such protograph designs, we have been able to limit the additional expansion nonoptimality to only about 0.1 to 0.2 dB if we apply our optimized expansion techniques such as PEG [67] and ACE [48] to build the full protograph code. While it is possible to push the protograph nonoptimality very close (perhaps arbitrarily close) to 0 dB, we have found that highly optimized protographs are too complex and produce intolerably large expansion nonoptimality when these protographs are expanded to codes of a few thousands or tens of thousands of bits. Thus, we generally look for protographs optimized under the constraint that they are also simple and small.

D. Performance of Actual Codes

The size-constrained nonoptimality, measured at a codeword error rate of $P_w = 10^{-4}$ is plotted on the horizontal axis of Fig. 10 for several families of LDPC codes [68], including two regular "G36" codes with variable nodes of degree 3, check nodes of degree 6, and code rate 1/2. Another key parameter of an LDPC code is the complexity of its decoder, measured in total number of edge metrics that must be computed. Again, this is primarily a function of code rate and block length: lower rate codes and longer block-length codes require more iterations. This dependence is fairly predictable, and when this component is removed, there is a remaining complexity variation due to the design of the code family. This is shown on the vertical axis of Fig. 10 in logarithmic units.

V. STANDARDIZATION OF CODES FOR DEEP SPACE

A. Standardization of Turbo Codes

Standardization of turbo codes [28] by the CCSDS was a remarkably efficient process. In part, this was because few proposals were involved; moreover, it was because there are relatively few parameters that must be

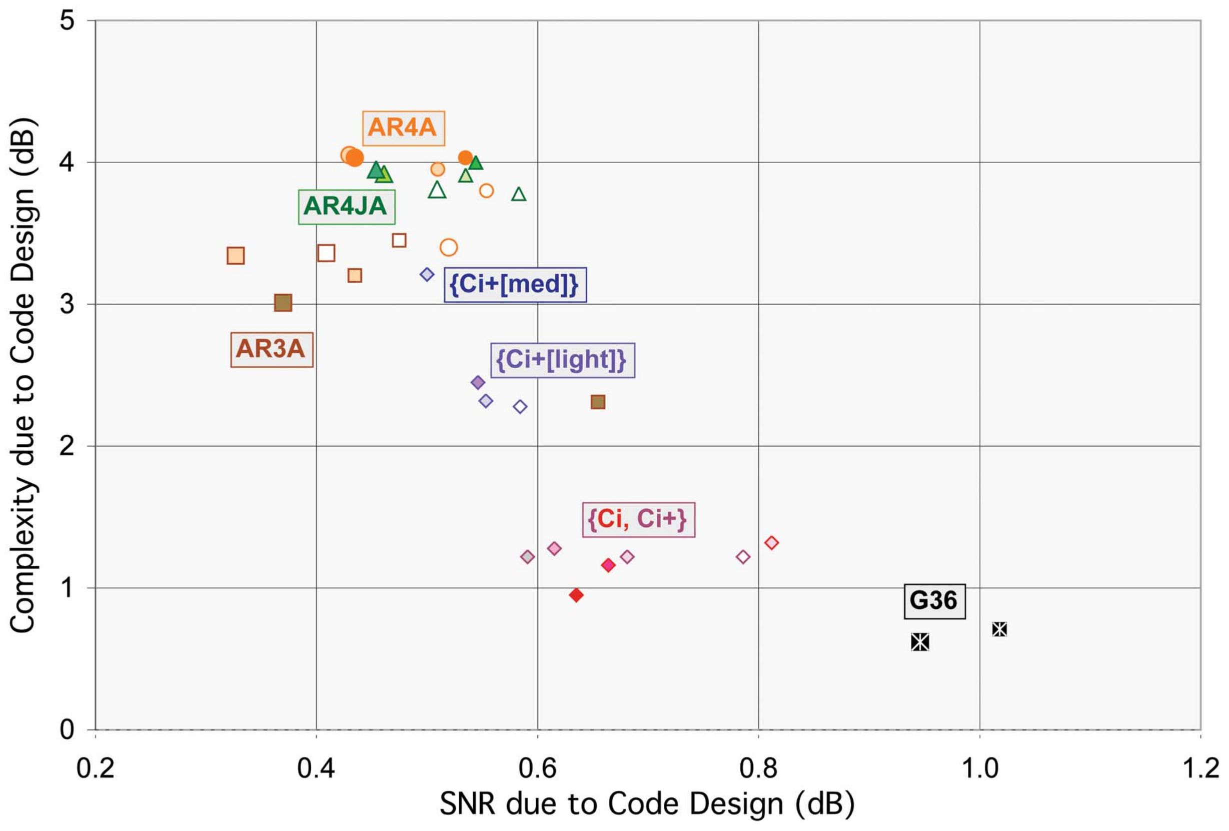


Fig. 10. ***Tradeoff between normalized decoder complexity and size-constrained nonoptimality among several LDPC code families at a codeword error rate of 10^{-4}.***

determined to define a turbo code. In fewer than six years from the initial discovery of turbo codes in late 1993, a CCSDS standard had been issued, describing the family of turbo codes depicted in Fig. 11.

When designing a turbo code, the first choice is in the constraint length of the constituent codes, which trades the decoder's computational complexity against required SNR. The CCSDS chose to use 16-state codes, in contrast to the 8-state codes chosen for third-generation wireless telephony and the IEEE 802.16 standard, among others. This was to save a fraction of a decibel in required SNR (an extremely valuable metric in deep space problems) at a cost of a factor of two in decoder complexity (a relatively cheap item in this setting). Aside from some research in asymmetric turbo codes [69], most turbo codes use two identical constituent convolutional codes with a primitive polynomial denominator of the specified constraint length, and numerator polynomials of the same degree. There are few enough such possibilities that a computer search can identify particularly good choices. The convolutional codes must be terminated in some way to prevent "end effects" from introducing low-weight codewords. The CCSDS choice was to independently terminate the interleaved and noninterleaved codes by driving each to encode to its zero state, a simple and sufficient solution [30], [70]. Finally, an interleaver must be chosen. The interleaver design primarily determines the level of the error floor, but despite a tremendous amount of research on this problem, good interleaver design remains something of an art. While a set of s-random interleavers [31] was proposed for their excellent performance, a Berrou-style [16] algorithmically specified interleaver was chosen for the standard. This allows a turbo encoder to generate the permutation with a logic circuit rather than by storing it as a large table in memory.

Performance is shown in Fig. 12 for two rate-1/6 turbo codes in comparison to the much more complex Cassini code of approximately the same rate (with two different interleaving depths, $I = 1, 5$).

B. Standardization of LDPC Codes

Standardization of LDPC codes by the CCSDS has proven challenging. In part, this is because the class of LDPC codes is very large, and their differences can be small. There is also a long list of desirable attributes [71], and no code is superior in all categories. The standardization process is ongoing, with a variety of candidates including serial concatenated turbo codes, the IRA code chosen by the second-generation Digital Video Broadcast standard (DVB-S2) [72], and regular and irregular LDPC codes.

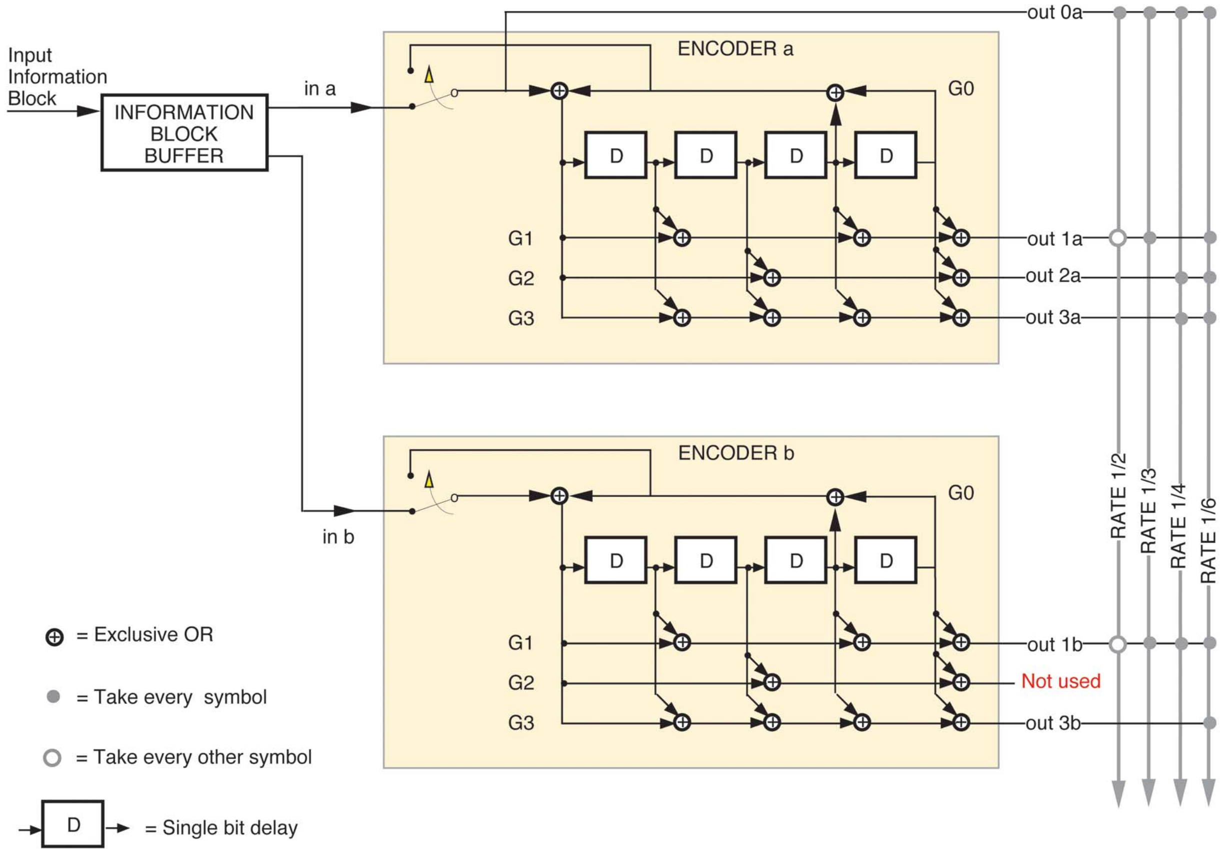


Fig. 11. ***The CCSDS turbo encoder.***

JPL has proposed a family of nine AR4JA LDPC codes with performances shown in Fig. 13 [29]. Also shown is the performance of code C_2, a regular LDPC code with variable nodes of degree 4, check nodes of degree 32, and rate $k/n = (511 \times 14 + 2)/(511 \times 16) = 0.8752$. A variant of code C_2 has also been proposed for CCSDS standardization [29].

VI. THE FUTURE AND OPEN QUESTIONS

Turbo codes were the first of the modern iteratively decoded codes to become practical. LDPC codes followed and have proven very versatile, but they have not replaced turbo codes, or even the traditional block and convolutional codes. LDPC codes are decoded on a parity check matrix, and this matrix grows larger as the code rate is decreased, making low-rate LDPC decoders more complex. In contrast, turbo codes are decoded on trellises, with one trellis section per information bit, corresponding to several code symbols. Hence turbo codes remain superior to LDPC codes at low rates. Iterative decoding, of either turbo or LDPC codes, remains complex relative to either Viterbi decoding of convolutional codes or to algebraic decoding techniques for Reed–Solomon and other block codes.

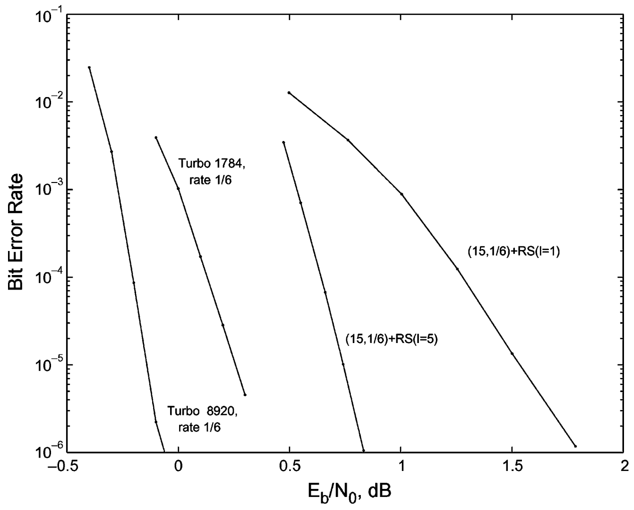


Fig. 12. ***Bit error rate curves for several codes with rates near 1/6: $k = 1784$ and $k = 8920$ turbo codes and the ($n = 255$, $k = 223$) Reed–Solomon code concatenated with a constraint length $\nu = 15$, rate 1/6 convolutional code.***

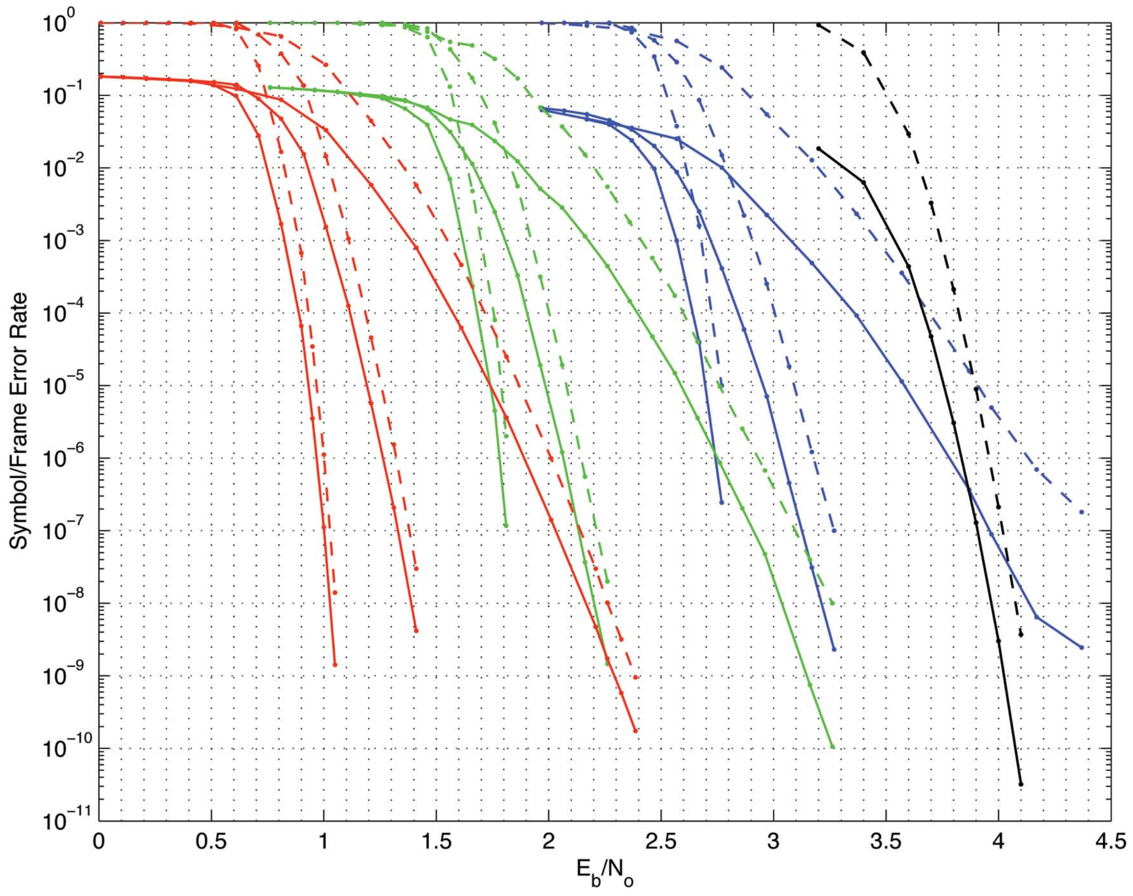


Fig. 13. ***Bit error rate (solid) and codeword error rate (dashed) for nine AR4JA codes and C_2, with code rates 1/2 (red), 2/3 (green), 4/5 (blue), and 0.8752 (black); and blocklengths $k = 16\,384, 4096, 1024$ (left to right in each group), and 7156 (code C_2).***

When decoding complexity is constrained, as it is in high-data-rate applications, the traditional codes remain unbeaten. It is unknown if there are fundamental reasons why these different niches require different coding solutions. It is quite possible that good LDPC codes based on generator matrices will be found, and that low complexity LDPC decoding algorithms will be discovered. If so, perhaps LDPC codes will eventually solve all coding problems. Decoder complexity in particular is unknown territory, with few theorems to guide the way. Analog LDPC decoders briefly appeared as a magic bullet to achieve extraordinary decoding speeds with relatively few transistors, but practical implementation has proven challenging [73], [74].

Iterative low-rate codes can provide reliable communication at $E_b/N_0 = 0$ dB and below. With a rate 1/6 code and binary phase-shift keying modulation, for example, $E_s/N_0 < -7.8$ dB, and few radio receivers can find and track the symbol timing in such a demanding situation. There is a need for improved receivers, perhaps using methods such as [75] and [76] to iteratively perform channel estimation and decoding.

An exciting trend in coding research today is in "universal codes" that are inherently optimal (or asymptotically so) for wide classes of problems, in the same way that the Lempel–Ziv data-compression algorithms are asymptotically optimal for any data source. Rateless erasure correcting codes [71], [77] are asymptotically optimal for channels with an arbitrary and unknown erasure rate. Perhaps error-correcting codes could also be designed analogously, so they adapt to the channel error statistics. ■

REFERENCES

[1] C. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, Jul./Oct. 1948, 623-656.

[2] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inf. Theory*, vol. IT-13, pp. 260–269, Apr. 1967.

[3] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Math.*, vol. 8, pp. 260–269, Jun. 1960.

[4] G. D. Forney, *Concatenated Codes*. Cambridge, MA: MIT Press, 1966.

[5] J. H. Yuen and Q. D. Vo, "In search of a 2-dB coding gain," JPL, TDA Prog. Rep. 42-83, Nov. 1985.

[6] O. Collins, F. Pollara, S. Dolinar, and J. Statman, "Wiring Viterbi decoders (splitting deBruijn graphs)," JPL, TDA Prog. Report 42-96, Feb. 1989.

[7] O. Collins, S. Dolinar, R. McEliece, and F. Pollara, "A VLSI decomposition of the deBruijn graph," *J. ACM*, vol. 39, no. 4, pp. 931–948, 1992.

[8] O. Collins, "Coding beyond the computational cutoff rate," Ph.D. dissertation, California Inst. of Technol., Pasadena, 1989.

[9] E. Paaske, "Improved decoding for a concatenated coding system recommended by CCSDS," *IEEE Trans. Commun.*, vol. COM-38, pp. 1138–1144, Aug. 1990.

[10] O. Collins and M. Hizlan, "Determinate state convolutional codes," *IEEE Trans. Commun.*, vol. 41, pp. 1785–1794, Dec. 1993.

[11] E. Paaske, "Alternative to NASA's concatenated coding system for the Galileo mission," *Proc. Inst. Elect. Eng. Commun.*, vol. 141, pp. 229–232, Aug. 1994.

[12] J. Statman, "Optimizing the Galileo space communication link," Jet Propulsion Laboratory, Pasadena, CA, TDA Prog. Rep. 42-116, Feb. 1994.

[13] S. Dolinar and M. Belongie, "Enhanced decoding for the Galileo S-band mission," Jet Propulsion Laboratory, Pasadena, CA, TDA Prog. Rep. 42-114, Aug. 1993.

[14] S. Dolinar and M. Belongie, "Enhanced decoding for the Galileo low-gain antenna mission: Viterbi redecoding with four decoding stages," Jet Propulsion Laboratory, Pasadena, CA, TDA Prog. Rep. 42-121, May 1995.

[15] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. IEEE Int. Conf. Commun.*, Geneva, Switzerland, May 1993, pp. 1064–1070.

[16] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-codes," *IEEE Trans. Commun.*, vol. 44, pp. 1261–1271, Oct. 1996.

[17] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inf. Theory*, vol. 42, pp. 429–445, Mar. 1996.

[18] P. Robertson, "Illuminating the structure of code and decoder of parallel concatenated recursive systematic (turbo) codes," in *Proc. Globecom Conf.*, 1994, pp. 1298–1303.

[19] S. Benedetto and G. Montorsi, "Unveiling turbo codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inf. Theory*, vol. 42, pp. 409–428, Mar. 1996.

[20] D. Divsalar and F. Pollara, "Turbo codes for deep-space communications," Jet Propulsion Laboratory, Pasadena, CA, TDA Prog. Rep. 42-120, Feb. 1995.

[21] S. Benedetto and G. Montorsi, "Average performance of parallel concatenated block codes," *Electron. Lett.*, vol. 31, pp. 156–158, Feb. 2, 1995.

[22] S. Benedetto and G. Montorsi, "Performance evaluation of turbo codes," *Electron. Lett.*, vol. 31, pp. 163–165, Feb. 2, 1995.

[23] D. Divsalar and F. Pollara, "Multiple turbo codes for deep-space communications," Jet Propulsion Laboratory, Pasadena, CA, TDA Prog. Rep. 42-121, May 1995.

[24] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "A soft-input soft-output APP module for iterative decoding of concatenated codes," *IEEE Commun. Lett.*, vol. 1, pp. 22–24, Jan. 1997.

[25] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding," *IEEE Trans. Inf. Theory*, vol. 44, pp. 909–926, May 1998.

[26] R. Gallager, "Low-density parity-check codes," Ph.D. dissertation, Massachusetts Inst. of Technol., Cambridge, Sep. 1960.

[27] J. Thorpe, "Low-density parity-check (LDPC) codes constructed from protographs," Jet Propulsion Laboratory, Pasadena, CA, IPN Prog. Rep. 42-154, Aug. 2003.

[28] *TM Synchronization and Channel Coding (131.0-B-1 Blue Book)*, Consultative Committee for Space Data Systems, Sep. 2003.

[29] *Low Density Parity Check Codes for Use in Near-Earth and Deep Space Applications (131.1-O-2 Orange Book)*, Consultative Committee for Space Data Systems, Sep. 2007.

[30] D. Divsalar and F. Pollara, "Turbo codes for PCS applications," in *Proc. IEEE Int. Conf. Commun.*, Seattle, WA, Jun. 1995.

[31] S. Dolinar and D. Divsalar, "Weight distributions for turbo codes using random and nonrandom permutations," JPL, TDA Prog. Rep. 42-122, Aug. 1995.

[32] J. Sun and O. Y. Takeshita, "Interleavers for turbo codes using permutation polynomials over integer rings," *IEEE Trans. Inf. Theory*, vol. 51, pp. 101–119, Jan. 2005.

[33] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inf. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.

[34] T. Miyauchi, K. Yamamoto, T. Yokokawa, M. Kan, Y. Mizutani, and M. Hattori, "High-performance programmable SISO decoder VLSI implementation for decoding turbo codes," in *Proc. IEEE Global Telecommunications Conf. (GLOBECOM)*, San Antonio, TX, Nov. 25–29, 2001, vol. 1, pp. 305–309.

[35] K. Andrews, V. Stanton, S. Dolinar, V. Chen, J. Berner, and F. Pollara, "Turbo-decoder implementation for the Deep Space Network," Jet Propulsion Laboratory, Pasadena, CA, IPN Prog. Rep. 42-148, Feb. 2002.

[36] A. Matache, S. Dolinar, and F. Pollara, "Stopping rules for turbo decoders," Jet Propulsion Laboratory, Pasadena, CA, TMO Prog. Rep. 42-142, Aug. 2000.

[37] J. B. Berner, K. S. Andrews, and S. H. Bryant, "Deep Space Network turbo decoder implementation," in *Proc. 2nd ESA Workshop Tracking Telemetry Command Syst. Space Applicat. (TTC 2001)*, Noordwijk, The Netherlands, Oct. 29–31, 2001.

[38] D. Mackay, "Good error correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, pp. 399–431, Mar. 1999.

[39] D. MacKay, S. Wilson, and M. Davey, "Comparison of constructions of irregular Gallager codes," *IEEE Trans. Commun.*, vol. 47, pp. 498–519, Oct. 1999.

[40] T. Richardson, "Multi-edge type LDPC codes," presented at the Workshop Honoring Prof. Bob McEliece on his 60th Birthday, Pasadena, CA, May 24–25, 2002.

[41] J. Thorpe, K. Andrews, and S. Dolinar, "Methodologies for designing LDPC codes using protographs and circulants," in *Proc. IEEE Int. Symp. Inf. Theory*, Jun. 2004, p. 238.

[42] Y. Kou, H. Tang, S. Lin, and K. Abdel-Ghaffar, "On circulant low density parity check codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Jun. 2002, p. 200.

[43] A. Abbasfar, D. Divsalar, and K. Yao, "Accumulate repeat accumulate codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Chicago, IL, Jun. 2004.

[44] D. Divsalar, S. Dolinar, and C. Jones, "Construction of protograph LDPC codes with linear minimum distance," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2006.

[45] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, 2nd ed. San Francisco, CA: Morgan Kaufmann, 1988.

[46] R. McEliece, E. Rodemich, and J. Cheng, "The turbo decision algorithm," in *Proc. 33rd Allerton Conf. Commun., Contr., Comput.*, 1995, pp. 366–379.

[47] X. Y. Hu, E. Eleftheriou, and D. M. Arnold, "Progressive edge-growth Tanner graphs," in *Proc. IEEE Global Telecommun. Conf.*, San Antonio, TX, Nov. 2001, pp. 995–1001.

[48] T. Tian, C. Jones, J. Villasenor, and R. D. Wesel, "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Trans. Commun.*, vol. 52, pp. 1242–1247, Aug. 2004.

[49] H. Jin, A. Khandekar, and R. McEliece, "Irregular repeat-accumulate codes," in *Proc. Int. Symp. Turbo Codes Related Topics*, Brest, France, Sep. 2000, pp. 1–8.

[50] T. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inf. Theory*, pp. 638–656, Feb. 2001.

[51] S. Lin, "Quasi-cyclic LDPC codes," CCSDS working group white paper, Oct. 2003.

[52] K. Andrews, S. Dolinar, and J. Thorpe, "Encoders for block-circulant LDPC codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Sep. 2005, pp. 2300–2304.

[53] C. Jones, E. Valles, M. Smith, and J. Villasenor, "Approximate-min* constraint node updating for LDPC code decoding," in *Proc. IEEE Military Commun. Conf. (MILCOM)*, Boston, MA, Oct. 2004.

[54] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, pp. 498–519, Feb. 2001.

[55] T. Richardson and V. Novichkov, "Node processors for use in parity check decoders," U.S. Patent 6 938 196, Aug. 2005.

[56] NASA, "NASA names new crew exploration vehicle Orion," press release, Aug. 2006.

[57] S. Dolinar, D. Divsalar, and F. Pollara, "Code performance as a function of block size," Jet Propulsion Laboratory, Pasadena, CA, TMO Prog. Rep. 42-133, May 1998.

[58] S. Dolinar, D. Divsalar, F. Pollara, and R. J. McEliece, "Transfer function bounds on the performance of turbo codes," Jet Propulsion Laboratory, Pasadena, CA, TDA Prog. Rep. 42-122, Aug. 1995.

[59] S. ten Brink, "Convergence of iterative decoding," *Electron. Lett.*, vol. 35, pp. 806–808, May 1999.

[60] H. El Gamal, "On the theory and applications of space-time and graph-based codes," Ph.D. dissertation, Univ. of Maryland, 1999.

[61] D. Divsalar, S. Dolinar, and F. Pollara, "Iterative turbo decoder analysis based on density evolution," *IEEE J. Sel. Areas Commun.*, vol. 19, pp. 891–907, May 2001.

[62] H. El Gamal and A. R. Hammons, Jr., "Analyzing the turbo decoder using the Gaussian approximation," *IEEE Trans. Inf. Theory*, vol. 47, pp. 671–686, Feb. 2001.

[63] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, pp. 599–618, Feb. 2001.

[64] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, pp. 69–637, Feb. 2001.
[65] T. Richardson, "Multi-edge type LDPC codes," presented at the Workshop Honoring Prof. Bob McEliece on his 60th Birthday, Pasadena, CA, May 2002.
[66] S. Y. Chung, "On the construction of some capacity-approaching coding schemes," Ph.D. dissertation, Massachusetts Inst. of Technology, Cambridge, Sep. 2000.
[67] X. Hu, E. Eleftheriou, and D. Arnold, "Irregular progressive edge-growth (PEG) Tanner graphs," in *Proc. IEEE Int. Symp. Inf. Theory*, Jun. 2002, p. 480.
[68] S. Dolinar and K. Andrews, "Performance and decoder complexity estimates for low-density parity-check (LDPC) code families," Jet Propulsion Laboratory, Pasadena, CA, IPN Prog. Rep. 42-168, Feb. 2007.
[69] P. Massey and D. Costello, "New developments in asymmetric turbo codes," in *Proc. Int. Symp. Turbo Codes*, Brest, Jun. 2000.
[70] K. Andrews, "Turbo codes and interleaver design," Ph.D. dissertation, Cornell Univ., Ithaca, NY, Aug. 1999.
[71] G. P. Calzolari, M. Chiani, F. Chiaraluce, R. Garello, and E. Paolini, "New requirements and trends for channel coding in future space missions," *Proc. IEEE*, vol. 95, no. 11, Nov. 2007.
[72] *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and Other Broadband Satellite Applications*, ETSI EN 302 307 V1.1.2, European Telecommunications Standards Institute, Jun. 2006.
[73] J. Hagenauer and M. Winklhofer, "The analog decoder," in *Proc. IEEE Int. Symp. Inf. Theory*, 1998, p. 145.
[74] H. Loeliger, F. Lustenberger, M. Helfenstein, and F. Tarkoy, "Decoding in analog VLSI," *IEEE Commun. Mag.*, pp. 99–101, Apr. 1999.
[75] J. Hamkins and D. Divsalar, "Coupled receiver-decoders for low rate turbo codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2003, p. 381.
[76] M. K. Simon, E. L. Valles, C. R. Jones, R. Wesel, and J. Villasenor, "Information-reduced carrier synchronization of BPSK and QPSK using soft decision feedback," in *Proc. 44th Allerton Conf. Commun., Contr., Comput.*, Sep. 2006.
[77] A. Shokrollahi, "Raptor codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Jun. 2004, p. 36.

ABOUT THE AUTHORS

Kenneth S. Andrews (Member, IEEE) received the B.S. degree in applied physics from the California Institute of Technology (Caltech), Pasadena, in 1990 and the Ph.D. degree in electrical engineering from Cornell University, Ithaca, NY, in 1999.

He is a member of the Communications Architectures and Research Section, Jet Propulsion Laboratory, Caltech. His current research work is in error-correcting code design and implementation, particularly for low-density parity-check and turbo codes.

Jon Hamkins (Student Member, IEEE) received the B.S. degree in electrical engineering from the California Institute of Technology (Caltech), Pasadena, in 1990 and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Illinois at Urbana-Champaign in 1993 and 1996, respectively.

Since then, he has been with the Jet Propulsion Laboratory, Caltech, where he is now Supervisor of the Information Processing Group.

Dariush Divsalar (Fellow, IEEE) received the Ph. D. degree in electrical engineering from the University of California, Los Angeles in 1978.

Since then, he has been with the Jet Propulsion Laboratory, California Institute of Technology, Pasadena, where he is a Principal Scientist.

Christopher R. Jones (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from the University of California, Los Angeles, in 1995, 1996, and 2003, respectively.

He was with Broadcom Corp. from 1997 to 2002. He has been with the Jet Propulsion Laboratory, California Institute of Technology, Pasadena, since 2004.

Sam Dolinar (Member, IEEE) received the S.B. degree in physics and the S.M., E.E., and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology (MIT), Cambridge, in the 1970s.

After four years with MIT Lincoln Laboratory, he joined the Jet Propulsion Laboratory, California Institute of Technology, Pasadena, in 1980. His primary research interests are error-correcting codes and data compression.

Fabrizio Pollara (Member, IEEE) received the M.S. and Ph.D. degrees in electrical engineering from the University of California, Los Angeles, in 1977 and 1982, respectively.

He joined the Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, in 1983, where he currently is Manager of the Communications Architectures and Research Section. He has contributed to the development of advanced communication systems for deep-space missions, including error-correcting codes.